МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«ВЕКТОРИЗАЦИЯ ВЫЧИСЛЕНИЙ»

студента 2 курса, 23202 группы

Пятанова Михаила Юрьевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
Владислав Александрович
Перепёлкин

Новосибирск 2024

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Изучение SIMD-расширений архитектуры x86/x86-64.

2. Изучение способов использования SIMD-расширений в программах на языке Си.

3. Получение навыков использования SIMD-расширений.

ОПИСАНИЕ РАБОТЫ

В первую очередь была написана программа без ручной векторизации (приложение 1), причем во многих местах был сделан ручной инлайнинг и количество обходов по циклу было сведено к минимуму. Здесь и далее компиляция производилась при оптимизации -O3. Данные оптимизационные преобразования были сохранены и в других вариантах программы. Корректность работы программы была проверена получением обратной матрицы:

1	2
3	4

Значение M	Результат			
10		-0,021	0,125	
		0,106	0,116	
1000		-1,916	0,963	
		1,440	-0,473	
100000		-1,999	1,000	
		1,499	-0,499	

Верный ответ:

-2	1
1,5	-0,5

Также были найдены обратные матрицы и большей размерности (приложение 1). Было измерено время работы программы для N=2048, M=10. Был получен наилучший результат: 196,17 с.

Далее была написана программа с ручной векторизацией, с применением встроенных функций компилятора, и её корректность также была проверена (приложение 2). Было измерено время работы программы для N=2048, M=10. Был получен наилучший результат: 41,54 с.

Третий вариант программы был написан с применением библиотеки BLAS. Код и проверки на корректность представлены в приложении 3. Было измерено время работы программы для N=2048, M=10. Был получен наилучший результат: 0,66 с.

В итоге получаем следующую таблицу времени работы алгоритма:

Вариант	Время, с.
Без ручной векторизации	196,17
Встроенные функции компилятора	41,54
BLAS	0,66

ЗАКЛЮЧЕНИЕ

Изучены SIMD-расширения архитектуры x86/x86-64, способы использования SIMD-расширений в программах на языке Си. Получены навыки использования SIMD-расширений.

Встроенные функции компилятора для работы с SIMD неожиданно показали эффективность в более чем 4 раза (в 4,78 раз) относительно реализации без ручной векторизации. Однако, самым эффективным по времени (в 296 раз) оказался алгоритм, реализованный с применением функций из библиотеки BLAS, что неудивительно. Хочется отметить тот факт, что подобные результаты вполне достижимы и вручную, при большей оптимизации (крайне сложной) программы (см. MIT 6.172 Performance Engineering of Software Systems, Fall 2018).

Приложение 1.

Команда для компиляции:
gcc lab7.c -o ex -O3

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

void sumMatrix(float** A, float** B, int N) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            A[i][j] += B[i][j];
        }
    }
}

void mulMatrix(float** res, float** A, float** B, int N) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            res[i][j] = 0;
            for (size_t k = 0; k < N; ++k) {
                res[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

float getNorm1Matrix(float** A, int N) {
    float max = __FLT_MIN__;
    for (size_t i = 0; i < N; ++i) {
        float norm = 0;
        for (size_t j = 0; j < N; ++j) {
            norm += fabsf(A[j][i]);
        }
        max = norm > max ? norm : max;
    }
    return max;
}

float getNormInfMatrix(float** A, int N) {
    float max = __FLT_MIN__;
    for (size_t i = 0; i < N; ++i) {
        float norm = 0;
```

```
        for (size_t j = 0; j < N; ++j) {
            norm += fabsf(A[i][j]);
        }
        max = norm > max ? norm : max;
    }
    return max;
}

void calculateB(float** A, float** B, int N, float scalar) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = i; j < N; ++j) {
            float c = A[j][i] * scalar;
            B[j][i] = A[i][j] * scalar;
            B[i][j] = c;
        }
    }
}

void calculateR(float** R, float** B, float** A, int N) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            R[i][j] = 0;
            for (size_t k = 0; k < N; ++k) {
                R[i][j] += B[i][k] * A[k][j];
            }
            R[i][j] = i == j ? 1 - R[i][j] : - R[i][j];
        }
    }
}

void equalMatrix(float** A, float** B, int N) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            A[i][j] = B[i][j];
        }
    }
}

void calculateReverseMatrix(float** res, float** A, int N, int M) {
    // Allocating memory for temporary matrices needed to calculate the result.
    float** temp1 = malloc(N * sizeof(float*));
    float** temp2 = malloc(N * sizeof(float*));
    float** R = malloc(N * sizeof(float*));
    float** B = malloc(N * sizeof(float*));
```

```
    for (size_t i = 0; i < N; ++i) {
        temp1[i] = malloc(N * sizeof(float));
        temp2[i] = malloc(N * sizeof(float));
        R[i] = malloc(N * sizeof(float));
        B[i] = malloc(N * sizeof(float));
    }
    // Getting time before calculation.
    struct timespec start, end;
    double time = 0;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);

    // Series computation.
    float scalar = 1 / (getNorm1Matrix(A, N) * getNormInfMatrix(A, N));
    calculateB(A, B, N, scalar);
    calculateR(R, B, A, N);
    // Many operations were inlined in order to get better time results.
    if (M == 1) {
        for (size_t i = 0; i < N; ++i) {
            for (size_t j = 0; j < N; ++j) {
                res[i][j] = i == j ? 1 : 0;
            }
        }
    } else {
        for (size_t i = 0; i < N; ++i) {
            for (size_t j = 0; j < N; ++j) {
                res[i][j] = i == j ? 1 + R[i][j] : R[i][j];
                temp1[i][j] = R[i][j];
                temp2[i][j] = R[i][j];
            }
        }
    }
    for (size_t i = 2; i < M; i++) {
        mulMatrix(temp2, temp1, R, N);
        sumMatrix(res, temp2, N);
        equalMatrix(temp1, temp2, N);
    }
    equalMatrix(temp2, res, N);
    mulMatrix(res, temp2, B, N);

    // Measure the time taken by the algo.
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    time = end.tv_sec - start.tv_sec + 0.000000001 * (end.tv_nsec - start.tv_sec);
    printf("Time taken: %lf\n", time);
```

```
    free(temp1);
    free(temp2);
    free(R);
    free(B);
}

int main() {
    int N = 2048;
    // Allocate memory for two matrices.
    float** A = malloc(N * sizeof(float*));
    float** res = malloc(N * sizeof(float*));
    for (size_t i = 0; i < N; ++i) {
        A[i] = malloc(N * sizeof(float));
        res[i] = malloc(N * sizeof(float));
    }
    // Fill A with random floats.
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i][j] = (float)rand() / RAND_MAX;
        }
    }
    // Calculate reversed A.
    calculateReverseMatrix(res, A, N, 10);
    // Print the result.
    // for (size_t i = 0; i < N; i++){
    //     for (size_t j = 0; j < N; j++) {
    //         printf("%f ", res[i][j]);
    //     }
    //     printf("\n");
    // }
    free(A);
    free(res);
    return 0;
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
-0.021567 0.124800
0.105760 0.116769
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
-1.916227 0.962941
1.440960 -0.473891
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
-1.999948 1.000055
1.499998 -0.499952
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
-0.500011 -0.500039 0.499876
-0.500005 0.999874 -0.500066
0.833295 -0.500003 0.166596
```

```
A[0][0] = 1; A[0][1] = 2; A[0][2] = 3;
A[1][0] = 4; A[1][1] = 6; A[1][2] = 6;
A[2][0] = 7; A[2][1] = 8; A[2][2] = 9;
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
Time taken: 198.043246
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
Time taken: 196.170021
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
Time taken: 197.409118
mikhail@mikhail:~/Desktop/evm/lab6$
```

Приложение 2.

Команда для компиляции:
gcc lab7.c -o ex -O3

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <xmmintrin.h>

#define VECTOR_SIZE 4
#define ALIGN 16

__m128 cyclicShiftLeft(__m128 vec) {
    __m128 shifted = _mm_shuffle_ps(vec, vec, _MM_SHUFFLE(0, 3, 2, 1));
    return shifted;
}

__m128 multyplyOnColumn(__m128 a, __m128 row1, __m128 row2, __m128 row3, __m128 row4)
{
    __m128 rowMul = a;
    rowMul = _mm_mul_ss(rowMul, row1);
    rowMul = cyclicShiftLeft(rowMul);
    rowMul = _mm_mul_ss(rowMul, row2);
    rowMul = cyclicShiftLeft(rowMul);
    rowMul = _mm_mul_ss(rowMul, row3);
    rowMul = cyclicShiftLeft(rowMul);
    rowMul = _mm_mul_ss(rowMul, row4);
    rowMul = cyclicShiftLeft(rowMul);
    return rowMul;
}

void sumMatrix(__m128** A, __m128** B, int N) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N / VECTOR_SIZE; ++j) {
            A[i][j] = _mm_add_ps(A[i][j], B[i][j]);
        }
    }
}

void mulMatrix(__m128** res, __m128** A, __m128** B, int N) {
    float temp[VECTOR_SIZE];
    float accum[VECTOR_SIZE];
    float acc = 0;
```

```
    int ind = 0;
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N / VECTOR_SIZE; ++j) {
            accum[0] = 0; accum[1] = 0; accum[2] = 0; accum[3] = 0;
            for (size_t k = 0; k < N / VECTOR_SIZE; k++) {
                __m128 rowMul = A[i][k];
                __m128 row1 = B[k*VECTOR_SIZE][j];
                __m128 row2 = B[k*VECTOR_SIZE + 1][j];
                __m128 row3 = B[k*VECTOR_SIZE + 2][j];
                __m128 row4 = B[k*VECTOR_SIZE + 3][j];
                for (size_t l = 0; l < VECTOR_SIZE; l++) {
                    rowMul = multyplyOnColumn(rowMul, row1, row2, row3, row4);
                    _mm_store_ps(temp, rowMul);
                    acc += temp[0];
                    acc += temp[1];
                    acc += temp[2];
                    acc += temp[3];
                    accum[ind] += acc;
                    ++ind;
                    row1 = cyclicShiftLeft(row1);
                    row2 = cyclicShiftLeft(row2);
                    row3 = cyclicShiftLeft(row3);
                    row4 = cyclicShiftLeft(row4);
                    rowMul = A[i][k];
                    acc = 0;
                }
                ind = 0;
            }
            res[i][j] = _mm_load_ps(accum);
        }
    }
}

void mulMatrixOld(float** res, float** A, float** B, int N) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            res[i][j] = 0;
            for (size_t k = 0; k < N; ++k) {
                res[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

```
float getNorm1Matrix(__m128** A, int N) {
    float max = __FLT_MIN__;
    float norms[VECTOR_SIZE];
    __m128 normSum;
    for (size_t i = 0; i < (N / VECTOR_SIZE); ++i) {
        // Set the four floats to the value 0.f.
        normSum = _mm_setzero_ps();
        for (size_t j = 0; j < N; ++j) {
            __m128 absVal = _mm_andnot_ps(_mm_set1_ps(-0.0f), A[j][i]);
            // r_i = a_i + b_i
            normSum = _mm_add_ps(normSum, absVal);
        }
        // norms[i] = normSum_i
        _mm_store_ps(norms, normSum);
        for (size_t j = 0; j < VECTOR_SIZE; ++j) {
            max = norms[j] > max ? norms[j] : max;
        }
    }
    return max;
}

float getNormInfMatrix(__m128** A, int N) {
    float max = __FLT_MIN__;
    float norms[VECTOR_SIZE];
    __m128 normSum;
    float norm;
    for (size_t i = 0; i < N; ++i) {
        norm = 0.f;
        normSum = _mm_setzero_ps();
        for (size_t j = 0; j < N / VECTOR_SIZE; ++j) {
            __m128 absVal = _mm_andnot_ps(_mm_set1_ps(-0.0f), A[i][j]);
            normSum = _mm_add_ps(normSum, absVal);
        }
        _mm_store_ps(norms, normSum);
        for (size_t j = 0; j < VECTOR_SIZE; ++j) {
            norm += norms[j];
        }
        max = norm > max ? norm : max;
    }
    return max;
}

void calculateB(__m128** A, __m128** B, int N, float scalar) {
    __m128 scalarVector = _mm_set1_ps(scalar);
```

```
    for (size_t i = 0; i < N; i += VECTOR_SIZE) {
        for (size_t j = 0; j < N; j += VECTOR_SIZE) {
            // Transpose a 4x4 tale.
            __m128 row0 = A[i + 0][j / VECTOR_SIZE];
            __m128 row1 = A[i + 1][j / VECTOR_SIZE];
            __m128 row2 = A[i + 2][j / VECTOR_SIZE];
            __m128 row3 = A[i + 3][j / VECTOR_SIZE];
            _MM_TRANSPOSE4_PS(row0, row1, row2, row3);
            // Apply scalar multiplication.
            row0 = _mm_mul_ps(row0, scalarVector);
            row1 = _mm_mul_ps(row1, scalarVector);
            row2 = _mm_mul_ps(row2, scalarVector);
            row3 = _mm_mul_ps(row3, scalarVector);
            // Store the transposed and scaled values in B.
            B[j + 0][i / VECTOR_SIZE] = row0;
            B[j + 1][i / VECTOR_SIZE] = row1;
            B[j + 2][i / VECTOR_SIZE] = row2;
            B[j + 3][i / VECTOR_SIZE] = row3;
        }
    }
}

void calculateR(__m128** R, __m128** B, __m128** A, int N) {
    float temp[VECTOR_SIZE];
    float accum[VECTOR_SIZE];
    float acc = 0;
    int ind = 0;
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N / VECTOR_SIZE; ++j) {
            accum[0] = 0; accum[1] = 0; accum[2] = 0; accum[3] = 0;
            for (size_t k = 0; k < N / VECTOR_SIZE; k++) {
                __m128 rowMul = B[i][k];
                __m128 row1 = A[k*VECTOR_SIZE][j];
                __m128 row2 = A[k*VECTOR_SIZE + 1][j];
                __m128 row3 = A[k*VECTOR_SIZE + 2][j];
                __m128 row4 = A[k*VECTOR_SIZE + 3][j];
                for (size_t l = 0; l < VECTOR_SIZE; l++) {
                    rowMul = multyplyOnColumn(rowMul, row1, row2, row3, row4);
                    _mm_store_ps(temp, rowMul);
                    acc += temp[0];
                    acc += temp[1];
                    acc += temp[2];
                    acc += temp[3];
                    accum[ind] += acc;
```

```
                    ++ind;
                    row1 = cyclicShiftLeft(row1);
                    row2 = cyclicShiftLeft(row2);
                    row3 = cyclicShiftLeft(row3);
                    row4 = cyclicShiftLeft(row4);
                    rowMul = B[i][k];
                    acc = 0;
                }
                ind = 0;
            }
            for (size_t k = 0; k < VECTOR_SIZE; ++k) {
                accum[k] = i == (j * VECTOR_SIZE + k) ? 1 - accum[k] : - accum[k];
            }
            R[i][j] = _mm_load_ps(accum);
        }
    }
}

void equalMatrix(__m128** A, __m128** B, int N) {
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N / VECTOR_SIZE; ++j) {
            A[i][j] = B[i][j];
        }
    }
}

void calculateReverseMatrix(__m128** res, __m128** A, int N, int M) {
    // Allocating memory for temporary matrices needed to calculate the result.
    __m128** temp1 = malloc(N * sizeof(__m128*));
    __m128** temp2 = malloc(N * sizeof(__m128*));
    __m128** R = malloc(N * sizeof(__m128*));
    __m128** B = malloc(N * sizeof(__m128*));
    for (size_t i = 0; i < N; ++i) {
        temp1[i] = _mm_malloc((N / VECTOR_SIZE) * sizeof(__m128), ALIGN);
        temp2[i] = _mm_malloc((N / VECTOR_SIZE) * sizeof(__m128), ALIGN);
        R[i] = _mm_malloc((N / VECTOR_SIZE) * sizeof(__m128), ALIGN);
        B[i] = _mm_malloc((N / VECTOR_SIZE) * sizeof(__m128), ALIGN);
    }
    // Getting time before calculation.
    struct timespec start, end;
    double time = 0;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);

    // Series computation.
```

```
    float scalar = 1 / (getNorm1Matrix(A, N) * getNormInfMatrix(A, N));
    calculateB(A, B, N, scalar);
    calculateR(R, B, A, N);
    // // Many operations were inlined in order to get better time results.
    __m128 row1 = _mm_set_ps(0.0f, 0.0f, 0.0f, 1.0f);
    __m128 row2 = _mm_set_ps(0.0f, 0.0f, 1.0f, 0.0f);
    __m128 row3 = _mm_set_ps(0.0f, 1.0f, 0.0f, 0.0f);
    __m128 row4 = _mm_set_ps(1.0f, 0.0f, 0.0f, 0.0f);
    if (M == 1) {
        for (size_t i = 0; i < N; ++i) {
            for (size_t j = 0; j < N / VECTOR_SIZE; ++j) {
                res[i][j] = _mm_setzero_ps();
                if (i == j * VECTOR_SIZE) {
                    res[i][j] = row1;
                    res[i + 1][j] = row2;
                    res[i + 2][j] = row3;
                    res[i + 3][j] = row4;
                    i += 4;
                }
            }
        }
    } else {
        for (size_t i = 0; i < N; ++i) {
            for (size_t j = 0; j < N / VECTOR_SIZE; ++j) {
                res[i][j] = R[i][j];
                temp1[i][j] = R[i][j];
                temp2[i][j] = R[i][j];
                if (i == j * VECTOR_SIZE) {
                    res[i][j] =  _mm_add_ps(res[i][j], row1);
                    res[i + 1][j] = R[i + 1][j];
                    res[i + 2][j] = R[i + 2][j];
                    res[i + 3][j] = R[i + 3][j];
                    res[i + 1][j] =  _mm_add_ps(res[i + 1][j], row2);
                    res[i + 2][j] =  _mm_add_ps(res[i + 2][j], row3);
                    res[i + 3][j] =  _mm_add_ps(res[i + 3][j], row4);
                    temp1[i + 1][j] = R[i + 1][j];
                    temp2[i + 1][j] = R[i + 1][j];
                    temp1[i + 2][j] = R[i + 2][j];
                    temp2[i + 2][j] = R[i + 2][j];
                    temp1[i + 3][j] = R[i + 3][j];
                    temp2[i + 3][j] = R[i + 3][j];
                    i += 4;
                }
            }
```

```
        }
    }
    for (size_t i = 2; i < M; i++) {
        mulMatrix(temp2, temp1, R, N);
        sumMatrix(res, temp2, N);
        equalMatrix(temp1, temp2, N);
    }
    equalMatrix(temp2, res, N);
    mulMatrix(res, temp2, B, N);

    // Measure the time taken by the algo.
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    time = end.tv_sec - start.tv_sec + 0.000000001 * (end.tv_nsec - start.tv_sec);
    printf("Time taken: %lf\n", time);

    free(temp1);
    free(temp2);
    free(R);
    free(B);
}

int main() {
    int N = 2048;
    if ((N % VECTOR_SIZE) != 0) {
        return 1;
    }
    // Allocate memory for two matrices.
    __m128** A = malloc(N * sizeof(__m128*));
    __m128** res = malloc(N * sizeof(__m128*));

    for (size_t i = 0; i < N; ++i) {
        A[i] = _mm_malloc((N / VECTOR_SIZE) * sizeof(__m128), ALIGN);
        res[i] = _mm_malloc((N / VECTOR_SIZE) * sizeof(__m128), ALIGN);
    }
    // Fill A with random floats.
    float temp[VECTOR_SIZE];
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N / VECTOR_SIZE; j++) {
            for (int k = 0; k < VECTOR_SIZE; k++) {
                temp[k] = (float)rand() / RAND_MAX;
            }
            // Load four single-precision floats. The address does not need to be 16-
byte aligned.
            // r_i = temp[i]
```

```
            A[i][j] = _mm_loadu_ps(temp);
        }
    }

    // for (size_t i = 0; i < N; i++) {
    //     for (size_t j = 0; j < N / VECTOR_SIZE; j++) {
    //         _mm_storeu_ps(temp, A[i][j]);
    //         for (size_t k = 0; k < VECTOR_SIZE; k++) {
    //             printf("%f ", temp[k]);
    //         }
    //     }
    //     printf("\n");
    // }

    // Calculate reversed A.
    calculateReverseMatrix(res, A, N, 10);

    //Print the result.
    // for (size_t i = 0; i < N; i++) {
    //     for (size_t j = 0; j < N / VECTOR_SIZE; j++) {
    //         _mm_storeu_ps(temp, res[i][j]);
    //         for (size_t k = 0; k < VECTOR_SIZE; k++) {
    //             printf("%f ", temp[k]);
    //         }
    //     }
    //     printf("\n");
    // }

    free(A);
    free(res);
    return 0;
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex -O3
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
3.000000 1.000000 2.000000 0.000000
3.000000 0.000000 1.000000 2.000000
4.000000 1.000000 2.000000 2.000000
0.000000 4.000000 3.000000 1.000000
Time taken: 0.166631
-0.593143 -1.995193 2.206406 -0.389948
-1.586258 -3.990438 4.212816 -0.379596
2.208664 5.003720 -5.385692 0.815630
-0.198921 0.999905 -0.598760 0.201671
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex -O3
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 3.000000 1.000000 2.000000 0.000000
 3.000000 0.000000 1.000000 2.000000
 4.000000 1.000000 2.000000 2.000000
 0.000000 4.000000 3.000000 1.000000
 Time taken: 1.090373
 -0.593143 -1.995193 2.206406 -0.389948
 -1.586258 -3.990438 4.212816 -0.379596
 2.208664 5.003720 -5.385692 0.815630
 -0.198921 0.999905 -0.598760 0.201671
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex -O3
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 3.000000 1.000000 2.000000 0.000000
 3.000000 0.000000 1.000000 2.000000
 4.000000 1.000000 2.000000 2.000000
 0.000000 4.000000 3.000000 1.000000
 Time taken: 9.305486
 -0.593143 -1.995193 2.206406 -0.389948
 -1.586258 -3.990438 4.212816 -0.379596
 2.208664 5.003720 -5.385692 0.815630
 -0.198921 0.999905 -0.598760 0.201671
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -o ex -O3
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 3.000000 1.000000 2.000000 0.000000 3.000000 0.000000 1.000000 2.000000
 4.000000 1.000000 2.000000 2.000000 0.000000 4.000000 3.000000 1.000000
 0.000000 1.000000 2.000000 1.000000 1.000000 3.000000 2.000000 4.000000
 2.000000 0.000000 2.000000 3.000000 2.000000 0.000000 4.000000 2.000000
 2.000000 3.000000 4.000000 2.000000 3.000000 1.000000 1.000000 2.000000
 4.000000 3.000000 1.000000 4.000000 4.000000 2.000000 3.000000 4.000000
 0.000000 0.000000 3.000000 1.000000 1.000000 0.000000 1.000000 3.000000
 2.000000 0.000000 1.000000 1.000000 0.000000 0.000000 4.000000 2.000000
 Time taken: 0.836683
 0.136877 0.161001 -0.228934 -0.182198 -0.140752 0.117753 0.203872 0.060087
 -0.179428 -0.144861 0.038131 -0.282750 0.360353 0.068351 -0.238836 0.344567
 0.026317 0.065009 -0.006708 0.068002 0.143482 -0.171667 0.110069 -0.037596
 -0.265511 0.073599 -0.190315 0.104996 -0.063709 0.191078 0.263513 -0.172459
 0.264269 -0.110704 0.217843 0.375358 -0.003553 -0.103622 -0.336390 -0.304641
 0.052365 0.130751 0.226332 0.120560 -0.049621 -0.076175 -0.170891 -0.232600
 0.000423 -0.093160 0.181071 0.193782 0.135704 -0.153120 -0.364445 0.207464
 -0.033130 -0.006971 -0.079527 -0.298324 -0.153388 0.209928 0.396000 0.133504
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7.c -O3 -o ex
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 Time taken: 41.537021
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 Time taken: 42.999482
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 Time taken: 44.007449
```

Приложение 3.

Команда для компиляции:
gcc lab7_blas.c -o ex -lblas -O3

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <cblas.h>

void sumMatrix(float* A, float* B, int N) {
    cblas_saxpy(N * N, 1.0, B, 1, A, 1);
}

void mulMatrix(float* res, float* A, float* B, int N) {
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0, A, N, B, N,
0.0, res, N);
}

float getNorm1Matrix(float* A, int N) {
    float max = __FLT_MIN__;
    for (size_t i = 0; i < N; ++i) {
        float norm = 0;
        for (size_t j = 0; j < N; ++j) {
            norm += fabsf(A[j * N + i]);
        }
        max = norm > max ? norm : max;
    }
    return max;
}

float getNormInfMatrix(float* A, int N) {
    float max = __FLT_MIN__;
    for (size_t i = 0; i < N; ++i) {
        float norm = 0;
        for (size_t j = 0; j < N; ++j) {
            norm += fabsf(A[i * N + j]);
        }
        max = norm > max ? norm : max;
    }
    return max;
}

void calculateB(float* A, float* B, float* Im, int N, float scalar) {
```

```
    cblas_sgemm(CblasRowMajor, CblasTrans, CblasNoTrans, N, N, N, scalar, A, N, Im,
N, 0.0, B, N);
}

void calculateR(float* R, float* B, float* A, float* Im, int N) {
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, -1.0, B, N, A, N,
1.0, R, N);
}

void equalMatrix(float* A, float* B, int N) {
    for (size_t i = 0; i < N * N; ++i) {
        A[i] = B[i];
    }
}

void calculateReverseMatrix(float* res, float* A, int N, int M) {
    // Allocating memory for temporary matrices needed to calculate the result.
    float* temp1 = malloc(N * N * sizeof(float));
    float* temp2 = malloc(N * N * sizeof(float));
    float* R = malloc(N * N * sizeof(float));
    float* B = malloc(N * N * sizeof(float));
    float* Im = malloc(N * N * sizeof(float));
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            Im[i * N + j] = i == j ? 1.f : 0.f;
            R[i * N + j] = i == j ? 1.f : 0.f;
        }
    }
    // Getting time before calculation.
    struct timespec start, end;
    double time = 0;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);

    // Series computation.
    float scalar = 1 / (getNorm1Matrix(A, N) * getNormInfMatrix(A, N));
    calculateB(A, B, Im, N, scalar);
    calculateR(R, B, A, Im, N);
    // // Many operations were inlined in order to get better time results.
    if (M == 1) {
        for (size_t i = 0; i < N * N; ++i) {
            res[i] = Im[i];
        }
    } else {
        for (size_t i = 0; i < N; ++i) {
```

```
            for (size_t j = 0; j < N; ++j) {
                res[i * N + j] = i == j ? 1 + R[i * N + j] : R[i * N + j];
                temp1[i * N + j] = R[i * N + j];
                temp2[i * N + j] = R[i * N + j];
            }
        }
    }
    for (size_t i = 2; i < M; ++i) {
        mulMatrix(temp2, temp1, R, N);
        sumMatrix(res, temp2, N);
        equalMatrix(temp1, temp2, N);
    }
    equalMatrix(temp2, res, N);
    mulMatrix(res, temp2, B, N);

    // Measure the time taken by the algo.
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    time = end.tv_sec - start.tv_sec + 0.000000001 * (end.tv_nsec - start.tv_sec);
    printf("Time taken: %lf\n", time);

    free(temp1);
    free(temp2);
    free(R);
    free(B);
}

int main() {
    int N = 2048;
    // Allocate memory for two matrices.
    float* A = malloc(N * N * sizeof(float));
    float* res = malloc(N * N * sizeof(float));
    // Fill A with random floats.
    for (int i = 0; i < N * N; i++) {
        A[i] = (float)rand() / RAND_MAX;
    }
    // for (size_t i = 0; i < N; i++) {
    //     for (size_t j = 0; j < N; j++) {
    //         printf("%f ", A[i * N + j]);
    //     }
    //     printf("\n");
    // }
    // Calculate reversed A.
    calculateReverseMatrix(res, A, N, 10);
    // Print the result.
```

```
    // for (size_t i = 0; i < N; i++) {
    //     for (size_t j = 0; j < N; j++) {
    //         printf("%f ", res[i * N + j]);
    //     }
    //     printf("\n");
    // }
    free(A);
    free(res);
    return 0;
}
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7_blas.c -o ex -lblas -O3
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
3.000000 1.000000 2.000000 0.000000
3.000000 0.000000 1.000000 2.000000
4.000000 1.000000 2.000000 2.000000
0.000000 4.000000 3.000000 1.000000
Time taken: 0.485921
-0.315130 -1.295557 1.450318 -0.299016
-1.036912 -2.607565 2.718139 -0.200424
1.478634 3.216382 -3.501984 0.544197
-0.297361 0.759235 -0.343765 0.165490
mikhail@mikhail:~/Desktop/evm/lab6$
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
3.000000 1.000000 2.000000 0.000000
3.000000 0.000000 1.000000 2.000000
4.000000 1.000000 2.000000 2.000000
0.000000 4.000000 3.000000 1.000000
Time taken: 0.347539
-0.593251 -1.995690 2.207198 -0.389764
-1.586403 -3.991401 4.214448 -0.379188
2.209727 5.005396 -5.386819 0.816661
-0.198819 1.000117 -0.598955 0.201695
mikhail@mikhail:~/Desktop/evm/lab6$
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7_blas.c -o ex -lblas -O3
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 3.000000 1.000000 2.000000 0.000000 3.000000 0.000000 1.000000 2.000000
 4.000000 1.000000 2.000000 2.000000 0.000000 4.000000 3.000000 1.000000
 0.000000 1.000000 2.000000 1.000000 1.000000 3.000000 2.000000 4.000000
 2.000000 0.000000 2.000000 3.000000 2.000000 0.000000 4.000000 2.000000
 2.000000 3.000000 4.000000 2.000000 3.000000 1.000000 1.000000 2.000000
 4.000000 3.000000 1.000000 4.000000 4.000000 2.000000 3.000000 4.000000
 0.000000 0.000000 3.000000 1.000000 1.000000 0.000000 1.000000 3.000000
 2.000000 0.000000 1.000000 1.000000 0.000000 0.000000 4.000000 2.000000
 Time taken: 0.552648
 0.134370 0.159704 -0.230511 -0.168356 -0.119437 0.098742 0.178171 0.051895
 -0.184938 -0.130334 0.014295 -0.281012 0.351530 0.070223 -0.212858 0.327845
 0.006642 0.076846 -0.042064 0.047954 0.153716 -0.166977 0.135205 -0.032466
 -0.270071 0.071729 -0.191920 0.114544 -0.040710 0.173976 0.237733 -0.175146
 0.244363 -0.106062 0.182216 0.315834 -0.031987 -0.068158 -0.265929 -0.276598
 0.044102 0.128700 0.215865 0.093099 -0.063342 -0.061707 -0.143425 -0.217240
 -0.003484 -0.095204 0.172300 0.164126 0.106402 -0.134492 -0.326155 0.214880
 0.012723 -0.042357 0.010646 -0.224703 -0.155794 0.160797 0.280392 0.106569
```

```
mikhail@mikhail:~/Desktop/evm/lab6$ gcc lab7_blas.c -o ex -lblas -O3
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 Time taken: 0.738730
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 Time taken: 0.911139
mikhail@mikhail:~/Desktop/evm/lab6$ ./ex
 Time taken: 0.662279
mikhail@mikhail:~/Desktop/evm/lab6$
```